

Compiler Project | Phase 1

02.04.2020

Ahmed Nabil (9)

Abdelrahman Alsayed Ahmed (24)

Mohamed Samy (40)

Yehia Elsayed (62)

Overview

In this project, it is required to build a compiler with its 3 first phases. In this phase, we have built the lexical analyzer generator. It is considered to be the first phase in the compiling process that translates the code into tokens which will help us in the second phase.

Data Structures

I. Building the NFA

- Unordered_map to save state transitions of every input
- Vector to save the states of the NFA
- Stack used in evaluation of the regular expression (transfer to postfix expression)

II. Building the DFA

- Vector of strings → final states of NFA, input tags, state tags[Name convention of NFA], DFA states.
- Vector of vectors of strings → DFA transitions, Partitioning.
- Map → Final states of DFA with its priority, Grouping of final states that are used in partitioning.
- Unordered_map → Final states of NFA with corresponding priorities.
- Vector of states[Implemented Class] → NFA transitions.

III. Analyzing the Test Code

- The same as NFA & DFA
- Vector of tokens[Implemented Class] → Tokens of the test code.

**** → : contains/holds ****

Algorithms & Techniques

I. Building the NFA

- Thompson's construction: a method of transforming regular expression to NFA.
- Postfix expression: used to sort the operations done in the regular expression.

II. Building the DFA

- The process of building DFA without minimization is implemented as described in the lecture by taking the initial state of NFA, get epsilon closure of it, this state will be the initial state of DFA and applying inputs to our new initial states with getting epsilon closure of the result. If the resulting state is defined for the first time, we put it into our transitions table of DFA and repeat previous steps on it and so on.
- Minimization's technique: Partitioning method . construct the initial partition π_0 of the set of states with two groups {final states , non-final states} , apply the partition method on π_0 , to check if each state is in its right state or need to be moved to another partition using some helping functions.the loop continues until that there is no change in the partition . for each group of states choose one state to be the representative for that group. remove the dead states and substitute each state in the transition table with its group representative.

III. Analyzing the Test Code

- Use maximal munch to detect the longest valid prefix in the input as a token and then go to the next character after the end of the last token
- Use panic mode recovery when no match is found by ignoring the first character and try to match from the next one.

Assumptions

- Range operation valid only between English characters (both capital or small) or between digits from 0 to 9
- And should the left character be less than the right character like b - d or 2 - 8.
- Any other way to describe range operation will be considered regular character
- When a rule in regular expression has an error then our code will ignore this rule.
 - "Input.txt" has the input program.
 - "Test.txt" has the rules.
 - "Output.txt" has the output stream.

Resultant of the minimal DFA

```

input tags
b o l e a n i t f 9 8 7 6 5 4 0 1 2 3 . E = ! > < s w h ; , ( ) { } + - * / z y x j g c d k m p q r u v Z Y X W J I H G F A B C D K L M N O P Q R S T U V

DFA final states with priority
Priority: 0 --> 42

Priority: 1 --> 33

Priority: 10 --> 12

Priority: 11 --> 13

Priority: 12 --> 14

Priority: 13 --> 15

Priority: 14 --> 16

Priority: 15 --> 17

Priority: 16 --> 18

Priority: 17 --> 1
Priority: 17 --> 2
Priority: 17 --> 3
Priority: 17 --> 4
Priority: 17 --> 5
Priority: 17 --> 10
Priority: 17 --> 19
Priority: 17 --> 20
Priority: 17 --> 21
Priority: 17 --> 27
Priority: 17 --> 28
Priority: 17 --> 29
Priority: 17 --> 31
Priority: 17 --> 34
Priority: 17 --> 35
Priority: 17 --> 37
Priority: 17 --> 38
Priority: 17 --> 39
Priority: 17 --> 40
Priority: 17 --> 41

Priority: 2 --> 30

Priority: 3 --> 6
Priority: 3 --> 25

Priority: 4 --> 9
Priority: 4 --> 23

Priority: 5 --> 7

Priority: 6 --> 32

Priority: 7 --> 36

Priority: 8 --> 22

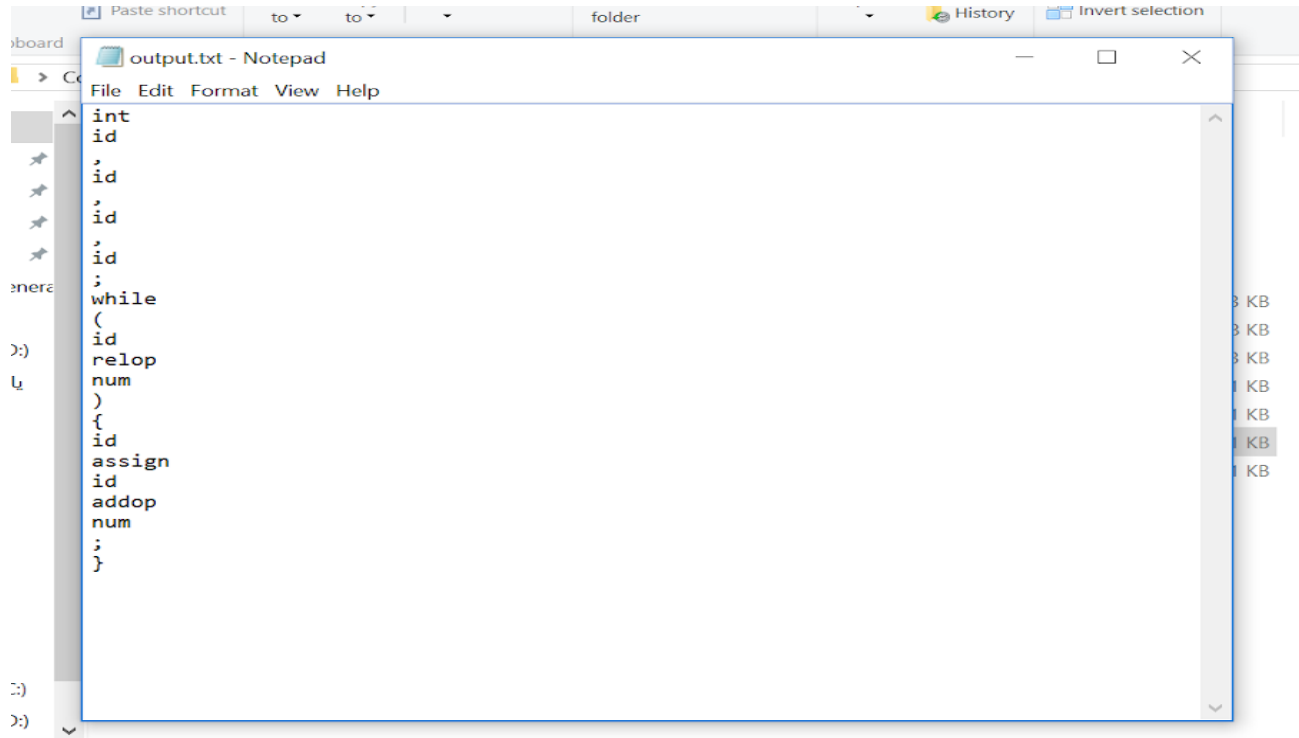
Priority: 9 --> 11

```

Input tags, Final states with rule priority



Resultant of the stream tokens for example program



The screenshot shows a Notepad window titled "output.txt - Notepad" with a menu bar (File, Edit, Format, View, Help). The text content is as follows:

```
int
id
,
id
,
id
,
id
;
while
(
id
relop
num
)
{
id
assign
id
addop
num
;
}
```

Resultant of the stream tokens for another program

<https://drive.google.com/open?id=1AixAAEd6wkaM6zUgWDsfxWPEZHpuAEqR>

Test Code of the other program

<https://drive.google.com/open?id=1tylFGqzhjVv1KkN88GkA21BiL52u7ft>

**** Rules are the same as the project's PDF ****

**** Results are provided with links due to the test code is very long to capture them as screenshots ****